A Work Project, presented as part of the requirements for the Award of a Master's degree in Finance from the Nova School of Business and Economics.

ONLINE ADVERTISING REVENUE FORECASTING: AN INTERPRETABLE DEEP LEARNING APPROACH

Max Würfel

Work project carried out under the supervision of:

Qiwei Han

04-01-2021

Abstract

This paper investigates whether publishers' Google AdSense online advertising revenues can be predicted from peekd's proprietary database using deep learning methodologies. Peekd is a Berlin (Germany) based data science company, which primarily provides eRetailers with sales and shopper intelligence. I find that using a single deep learning model, AdSense revenues can be predicted across publishers. Additionally, using unsupervised clustering, publishers were grouped and related time series were fed as covariates when making predictions. No performance improvement was found in relation with this technique. Finally, I find that in the short-term, publishers' AdSense revenues embed similar temporal patterns as web traffic.

Keywords: Deep Learning, Time Series Forecasting, Interpretability, Online Marketing.

Acknowledgments: I gratefully acknowledge the help I received from my supervisor Qiwei Han. I also wish to thank Maximilian Kaiser and Daniel Schmeh from peekd for providing me with the data to be used in my work project as well as their continued support.

This work used infrastructure and resources funded by Fundação para a Ciência e a Tecnologia (UID/ECO/00124/2013, UID/ECO/00124/2019 and Social Sciences DataLab, Project 22209), POR Lisboa (LISBOA-01-0145-FEDER-007722 and Social Sciences DataLab, Project 22209) and POR Norte (Social Sciences DataLab, Project 22209).

1 Introduction

This study was conducted in collaboration with peekd, a Berlin (Germany) based data science company. Peekd offers eRetailers the opportunity to receive benchmark analytics based on its proprietary database. Customers in turn will connect their Google Analytics account to peekd's infrastructure. For some clients, the accessible data encompasses Google AdSense revenues, a service that provides publishers with the opportunity to monetize their website traffic by hosting ads on their website and advertisers to serve their ads to a targeted audience. This study intends to first, showcase how peekd can take advantage of its proprietary data in generating accurate AdSense revenue forecasts. Second, reinforce the applicability of interpretable deep learning techniques to business settings. And finally, provide insight into the drivers of AdSense revenues for publishers.

Peekd's business model relies on the supply of analytics and visualization tools to their customers. Thus, a model that can provide AdSense revenue forecasts for peekd's clients, constitutes a new product, which could be deployed as part of the firm's service offering. Generally, peekd might offer distinctive value to its customers in three ways.

First, peekd is an experienced data science player who has successfully implemented machine learning models across a set of applications. For most publishers, it would unquestionably be less expensive to outsource the development of any deep learning model to a specialized company, while achieving more accurate results in a shorter time (Edlich et al. 2019). Meanwhile, by deploying any model for multiple firms, peekd could benefit from scale economics.

Second, evidence suggests that peekd's access to cross-sectional data might allow for superior model performance. Amazon recently released DeepAR, a time series forecasting model, which illustrates how deep learning models, trained on cross-sectional units, can outperform models like ARIMA trained on a single unit's time series. Besides, the authors showcased that the model performance is increasing in the number of cross-sectional units and that the effect is reinforced when the individual time series are correlated (Flunkert, Salinas, and Gasthaus 2017). For instance, web-traffic between two newspapers might be positively correlated in the short-term, due to breaking news, and negatively correlated in the long-term, due to competing sources of similar information, causing correlated AdSense revenues. Hence, there might be a significant

performance boost from training the model on peekd's panel data.

Finally, I aim to take a step further in levering access to the database. Given multiple related online advertisement revenue time series, like those of competitors, the correlated time series might act as additional features in the prediction problem. Whether this technique can further improve model performance, needs to be confirmed by the end of this study.

Next to its business relevance, the prediction of AdSense revenues for publishers constitutes a novel problem, which has not been investigated before. I deploy an interpretable deep learning approach to provide insight into the drivers of AdSense revenues.

The remainder of this paper is organized as follows. First, Sec. 1 will continue by defining the prediction problem, explaining the functioning of the Google AdSense network, and highlighting the advantages of deep learning compared to statistical methods. Sec. 2 is concerned with related work on Google AdSense and Deep Learning for time-series forecasting. The methodology is discussed in Sec. 3, while results are analyzed in Sec. 4. Sec. 5 concludes.

1.1 Problem Definition

Given the historical sequence of a publisher's AdSense revenues, accompanying time-series data and categorical information, the goal is to forecast the AdSense revenues over multiple days. More formally, the multi-horizon prediction problem is defined as:

$$y_{i,t+\tau} = f(y_{i,t-k:t}, u_{i,t-k:t+\tau}, x_{i,t-k:t}, s_i) \tag{1}$$

Where $\tau_{max} = 30$ and $k_{max} = 89$, such that the intention is to output daily revenue predictions over roughly the next month based on the time series from the last quarter. Inputs can be divided into four distinct groups, where i always represents an individual publisher. The first group is composed of the target variable $y_{i,t-k:t} = \{y_{i,t-k}, \ldots, y_{i,t}\}$, which denotes the historical AdSense revenue. The second group are covariates with known future observations $u_{i,t-k:t+\tau} = \{u_{i,t-k}, \ldots, u_{i,t+\tau}\}$, like the day of a week an observation will occur. The next group is made up by inputs $x_{i,t-k:t} = \{x_{i,t-k}, \ldots, x_{i,t}\}$, which are covariates, that are unknown over the prediction time horizon, like page views. Finally, s_i describes static features such as the home country of the publisher.

1.2 Google AdSense

Google AdSense is an advertisement network, through which publishers can monetize their website traffic by serving ads on their website. Hereby, the publisher‘s revenue is usually generated on a cost-per-click (CPC) basis. CPC is the amount an advertiser will pay for a click on its ad, whereby about 70% of the revenues will be distributed to the publisher (Google 2020). Advertisers, on the other hand, have the opportunity to reach a specific target audience. The ads, which may be rendered on a publisher's website are automatically selected by Google based on three criteria. First, contextual targeting, which refers to the matching between a website's contents and potential ads. Second, placement targeting concerns the preferred consideration of advertisers who registered their intention to advertise on specific websites. Third, personalized targeting involves the analysis of the audiences' demographic data. This includes geographical location and device, but also specific user characteristics, like "sports enthusiast" (Google 2020). To participate in the Google AdSense network, any publisher only needs to reserve space on their website and inject a code snippet in their website's frontend. When a user visits the page, the publisher will send audience information, like the origin and device of the user to AdSense. Subsequently, suiting ads and the corresponding advertisers are selected by Google following the criteria discussed above. Next, the selected advertisers can participate in an auction to show their ad. In the auction, the ad's rank is not only determined by the CPC bid but also the ad quality score, which tries to capture the expected user experience and the likelihood that the customer will click the ad (Google 2020). Usually, advertisers do not submit bids for single websites but for keywords, where a keyword is associated with multiple websites. Any keyword represents the contents of the website and therefore also the characteristics of the target audience (Wang and Chen 2012). The winning bidder finally sends a script to the publisher's website, where the contents are rendered. Thus, the ads displayed on any publisher's website, the number of users clicking the ad, and eventually, the revenues earned will vary over time.

1.3 Deep Learning for Time Series Forecasting

Deep Learning, while gaining its original popularity from Computer Vision and Natural Language Processing (NLP) tasks, has been frequently applied to make time-series forecasts. Driven

by the increasing data availability and computing power, deep learning allows to learn temporal patterns without the need for extensive preprocessing, assumptions about the underlying distribution, and exhaustive feature engineering (Lim and Zohren 2020).

Nevertheless, statistical models like ARIMA continue to perform distinctively better on univariate problems, especially for short time series. In the case at hand, the problem is multivariate and over a thousand daily observations per time series are available. With this length of time series data, it has been shown that deep learning methodologies will outperform statistical models reliably (Cerqueira, Torgo, and Soares 2019). A more practical reason in favor of deep learning methodologies is the fact that only a single model needs to be fit, which can make predictions across all publishers i in the dataset. Generally, most statistical models show diminishing performance in high-dimensional multivariate problems, where deep learning models start to benefit from their ability to fit non-linear, complex functions. Further, to make multi-step predictions, models like ARIMA can solely be used recursively, making one-step-ahead predictions, while machine learning models allow for more advanced output strategies.

Finally, when compared to statistical methodologies, there is no need to select from parametric and non-parametric models. For many statistical models, in turn, the data needs to be parametric, in particular stationery, such that mean and variance are constant over time. Contrarily, deep learning models usually learn to deal with seasonalities and trends independently.

Historically, a caveat to deep learning methodologies has been the lack of interpretability. But as new deep learning methodologies emerge, efforts have been made to combat this issue (Lim and Zohren 2020). In business applications like the one at hand, interpretability is of utmost importance. This study aims to solely apply deep learning methodologies to produce forecasts, with special attention on the interpretability of the results.

2 Related work

2.1 Google AdSense Revenues

To the best of my knowledge, no research exists, solely concerned with the AdSense revenue prediction for publishers. On the web, there are several AdSense revenue calculators available and even Google offers a tool, which intends to give a rough estimate of expected online

advertisement revenues. Usually, any tool will ask the publisher to impute their website category, region, and expected website traffic. Notably, the selection of inputs made already gives a first indication of the underlying dynamics of AdSense revenues, which will be discussed below. Clearly, none of the tools leverage historical information as intended in this study.

Adapted from Wang and Chen (2012), every publisher will try to maximize its profit $P = CPC \times clicks = CPC \times impressions \times CTR$ where CTR is the click-through-rate. Even though page views and impressions are technically not the same, they are strongly related. While the page views measure the number of times a user visits a webpage, the impressions measure the frequency a specific ad is viewed. Naturally, this relationship is less variable than the other determinants in the profit-maximizing objective, such that page views and ad impressions will be treated interchangeably. Thus, the generated AdSense revenue for publishers will mainly depend on three factors.

First, website traffic, as the volume of visitors on a website changes, the number of possible ad impressions and clicks will adapt accordingly. Multiple studies concern themselves with the prediction of website traffic, mostly deploying statistical methods. According to Li and Moore (2008), web traffic time series often embed strong weekly correlation, while milder yearly dependencies prevail. Further website traffic is often assumed to be stationary in the short-term but tends to exhibit trends in the long-term (Li and Moore 2008). Such patterns might consequently also be reflected in the AdSense revenue development.

Second, the CTR will determine the number of website visitors, which actually click the ad and consequently generate revenues. A complete stream of literature is concerned with CTR probability prediction (McMahan et al. 2013). Indeed, almost any advertisement network will use the expected CTR in the bidding process. In Google AdSense, the expected CTR is embedded in the quality score from Sec. 1.2. Unfortunately, most studies are concerned with the quick binary prediction of whether a given visitor will click an ad or not, which only yields limited insight into the dynamic of a publisher's online advertisement revenues (Zhou et al. 2018).

Finally, the CPC will determine the compensation for a publisher. Evidently, the determinants of the CPC are the maximum bids submitted in the relevant auction. Usually, the competition in the auction will depend on the relevant keyword and in turn the expected CTR, while ultimately

also on the individual advertiser's conversion rate and average item value (Google 2020).

Rather than a single visitor's CTR or a single CPC, decisive key ratios in predicting a publisher's AdSense revenue are the average CTR and CPC of its website, across all ads shown over a specific timeframe. In an effort to determine the expected value of the dollar spent on online advertisements, research exists aiming to predict CTR and CPC on keyword-level (Wang and Chen 2012). Given, that the contents of any publisher's website will be the main determinants of the keywords assigned by Google towards that page, keywords are unlikely to change frequently. Similarly, the target audience of a website can be expected to hardly change while relying on the continued supply of tailored content. The target audience, in turn, is strongly correlated to buying power and as such also the CPC. Hence, the contents of a website and the corresponding audience, through the associated keywords, might yield insight into the average CTR and CPC of a publisher. Even more, websites grouped by their contents can be expected to share keywords and thus also CTR and CPC rates.

It also follows that publishers seem to have little tools at their disposal to impact their online advertisement revenues. Quick and abrupt changes might be caused by the positioning of the ads, the number of the ads, and the media type of the ads on the website, but not by changes to content or audience. Significantly more power in the development of AdSense revenues might be with the advertisers, which set the CPC based on their individual objectives. Advertisers might decide to suddenly increase their advertising budget and thereby drive up demand in an attempt to promote a certain product (Google 2020). Subsequently, AdSense revenues might embed patterns over which publishers have little control, but which notably influence their ad income, while also reducing predictability.

When trying to predict CTR and CPC, Wang and Chen (2012) make an important distinction concerning the feature space. On the one hand, there are contextual features, which represent information on the context an ad is shown in, e.g the number of words used in the ad title or demographic information associated with a specific user. On the other hand, there are historical features, which encompass time series like CTR, CPC, and page views. In the study at hand, only historical features will be levered to make predictions, as contextual features are naturally unavailable over a multi-step horizon.

Finally, a question remains as to why there is so little research concerned with the forecasting of AdSense revenues for publishers. First, as described above, for publishers facing deteriorating AdSense revenues, there is the lack of corrective action in the short-term, as content and audience are fixed. Second, the access to AdSense revenue data is proprietary. Typically, no publisher will share its AdSense revenue data with its competitors, thus, peekd's database might yield a unique opportunity to investigate the topic.

To conclude, there are three main determinants of AdSense revenues, which are page views, CTR, and CPC. Further, there is evidence indicating that for any publisher, CTR and CPC, through the associated keywords, depend on its website's audience and content. As neither of these can be expected to change frequently, also CTR and CPC are expected to stay fixed in the short-term. Hence, web traffic might account for most short-term movement in AdSense revenues. Additionally, a great portion of the variability in ad revenues, and specifically CPC, might be driven by advertisers' actions and thus, lie outside of the control of publishers.

2.2 Deep Learning for Time Series Forecasting

2.2.1 Model Architectures

There are two main types of deep learning architectures frequently applied to generate time-series forecasts. First, there are Convolutional Neural Networks (CNNs) from Computer Vision and second, Recurrent Neural Networks (RNNs) from Natural Language Processing (NLP).

To apply CNNs to time-series prediction problems, multiple convolutional layers are usually stacked on top of each other like in Borovykh, Bohte, and Oosterlee (2017). In addition, regular convolutions are swapped for causal convolutions, which ensure that any prediction $p(y_{t+1}|y_{t-k}, ..., y_t)$ can only depend on historical time steps up until y_t (Oord et al. 2016). As in Computer Vision, CNNs extract temporal features by shifting a single kernel, with the same weights, along the time series. Hereby, the lookback window (receptive field) is the number of historic time steps a single kernel attends when generating outputs. Historically, the biggest challenge in time series forecasting used to be the consideration of long time series intervals and lookback windows, causing computational challenges (Lim and Zohren 2020). Dilated convolutions constitute a possible solution to this problem, allowing for increased lookback

window width while holding the number of parameters constant (Yu and Koltun 2015). This is achieved by the dilated convolution simply skipping certain input time steps according to the dilation rate. Notably, the input layer usually does not incorporate any dilation, while convolutional layers further up in the structure use increasing dilation rates.

A widely used adoption of a dilated CNN is WaveNet, a neural network architecture originally intended to generate audio signals by Oord et al. (2016), and frequently adopted to produce time-series forecasts (Borovykh, Bohte, and Oosterlee 2017). On top of the dilated convolutions, there are two other especially interesting features of the WaveNet architecture, which have both been adapted for time series prediction. First, the usage of gated activations, and second, the usage of skip and residual connections. In gated activations, the output from a convolutional layer is split into two branches and later recombined by element-wise multiplication. Thereby, one branch can be interpreted as a learned filter, while the other branch, which is sigmoid activated, can be thought of as the gate. Gated activations have been seen to outperform more standard activation such as the rectified linear unit (ReLU), not only in speech generation but also in time-series tasks. In particular, a gated linear unit (GLU) can be used as gated activation in time-series prediction (Dauphin et al. 2017):

$$GLU(x) = sigmoid(W_1 x + b_1) \odot (W_2 x + b_2) \tag{2}$$

where the gating mechanism allows for the selection of important signals and suppression of noise (Lim et al. 2020).

Second, WaveNet features skip and residual connections, where I will focus on the residual connections. Introduced by He et al. (2016), a residual block is only learning the residual function and allows a network block's input x to skip the non-linear transformation by being subsequently recombined with the residual block's output. Given a true distribution $H(x)$ to be learned, a residual block with input x is given as:

$$H(x) = R(x) + x \tag{3}$$

where $R(x)$ is the residual learned by the network block, which is added to the unaltered input to learn the desired mapping. In some situations, it might be easier to learn the residual function

instead of the true underlying mapping. Also, this architecture makes it significantly easier to learn an identity mapping by solely setting the residual to zero (He et al. 2016). Both gated activations and residual skip connections are very relevant to time series prediction problems. Since time series are usually made up of different components, like autoregressive relations, trends, seasonalities, and hard-to-spot trajectories, some easy-to-spot, autoregressive relations might not require as much non-linear processing as other, more complex feature representations. The second class of model architectures successfully utilized in time series forecasting are RNNs. Any RNN positions multiple cells on a directed graph, where the weights are shared across all cells. Further, each cell contains an internal memory state, which is continuously updated based upon new observations (Rumelhart, Hinton, and Williams 1986). To combat the historically prevailing issue of fading transmission power, the Long Term Short Term Memory cell (LSTM) has been introduced, which updates an internal cell state in addition to the hidden state (Hochreiter and Schmidhuber 1997). This is relevant in time series forecasting, as historical information can influence predictions far in the future.

In its original form, LSTMs perform well when the target is to produce a single output, but not when the output is a sequence itself, like in multi-horizon time series forecasting. Eventually, Sutskever, Vinyals, and Le (2014) introduced a Sequence to Sequence (Seq2Seq) network architecture, which levers two distinct LSTMs and can be thought of as a conditional autoregressive model. The first network encodes the input sequence into a fixed-length vector, while the second one decodes the encoded state to generate predictions, one step at a time. Thereby, the predictions are recursively fed into the next decoder cell to produce the next prediction.

Another important addition to Seq2Seq architectures but also the main building block of Transformer architectures, has been the introduction of attention layers (Bahdanau, Cho, and Bengio 2014). Thereby, the attention mechanism can effectively transfer information from relevant past time steps to the corresponding output step. In a Seq2Seq architecture, for instance, this means allowing the hidden states of a very early encoder cell to directly impact a decoder cell many steps ahead. In a time series context, this could be interpreted as the ability to attend to certain special events like Christmas for sales forecasting. Adopted from Vaswani et al. (2017) any

attention layer computes a context vector h_t at time step t as:

$$h_t = \sum_{\tau=0}^{k} \alpha(q_t, k_\tau) v_\tau \tag{4}$$

where k_τ is the key and v_τ is the value, which in Bahdanau, Cho, and Bengio (2014) would both be represented by the hidden encoder states. And where q_t is the query, which in Bahdanau, Cho, and Bengio (2014) would be the past decoder cell's hidden state. Finally, $\alpha(q_t, k_\tau)$ are the attention weights generated for time step τ at time step t (Lim and Zohren 2020).

More recently, transformer architectures have also been deployed in time series forecasting problems (Lim et al. 2020). Hereby, an adaption of the original attention mechanism namely self-attention is used, where query, key, and value are all computed from the same initial vector and the dot scalar product is used to generate attention weights. In transformers, multi-head-self-attention (MSA) layers deploy multiple attention mechanisms called heads, all operating on the same values, hence attending the same input sequence. This architecture has recently been shown to also improve time-series forecasts (Li et al. 2019). In particular, each head of the MSA layer has been found to capture different temporal structures, like autoregressive relations or seasonalities.

2.2.2 Multi-Step Forecasting strategies

In many applications, it is worthwhile to generate a sequence of forecasts over a specified time horizon. Multiple strategies exist to make multi-step predictions.

First, a distinct model can be developed for each timestep, each generating its own one-step-ahead prediction. As the number of models is linearly increasing in the forecasting horizon, this method is infeasible for longer time frames. Second, it is possible to deploy a recursive (autoregressive) strategy to make multi-step predictions. Thereby, a one-step-ahead model is trained and at runtime, the one-step-ahead predictions are appended to the input vector. The caveat of this prediction strategy is that when using multi-variate inputs, a prediction needs to be made for each input series independently to be able to generate one-step-ahead predictions. Additionally, errors made in early predictions can accumulate over the prediction horizon (Lim and Zohren 2020). Finally, it is possible to utilize a multi-output strategy, where a single

model outputs multiple predictions at the same time. While only a single model needs to be trained, this strategy usually requires the usage of more complex model architectures to successfully make predictions. To directly generate output vectors, either basic models like RNNs can be manipulated to output a fixed-length vector instead of a single prediction. Or, a more sophisticated approach levers encoder-decoder architectures, like Seq2Seq (Sec. 2.2.1).

3 Methodology

3.1 Data and Preprocessing

The data for this study was provided by peekd and is part of their proprietary database. From hereafter, the terms publishers and domains are used interchangeably, as, within the dataset, a domain will uniquely identify a publisher. For all publishers in the database, daily AdSense revenues from the 01.01.2018 until the 30.10.2020 are available. Additionally, time series for ad impressions, ad clicks, page views, and bounces are available (data dictionary in Appx. B). All daily observations are available for three device categories, namely desktop, mobile, and tablet. For the purpose of this study, only the desktop data is considered as it captures the biggest fraction of the total revenue generated. Combining all the device categories as a total is not a viable option, as ad clicks and impressions from different device categories convert into AdSense revenues at significantly different rates.

The dataset encompasses data on over 400 domains, from whom only a notably smaller subset is deployed in this study. First, a geographical subset of the domains is taken. In line with Sec. 2.1, the potential customer value differs across regions, such that the AdSense revenue transmitters CTR and CPC might behave similarly within a region. Notably, the origin of a publisher will not entirely match the origin of its users, but it provides a reasonable proxy. In particular, the dataset has been reduced to incorporate only domains from fifteen developed countries (Appx. A). Another reason for deploying a regional subset of the data might be that culturally close countries are likely to experience similar trends and patterns in their website traffic.

Equivalently, a subset of the data could have been selected based upon the websites' category. In particular, categorically grouped domains might offer similar contents and be associated with similar keywords. In an attempt to keep a reasonable dataset size while also allowing for rigorous

data cleaning, no such reduction has been performed.

The central concern with the provided dataset is the high frequency of zero AdSense revenue observations as well as frequent and abrupt shifts to the overall level of generated revenues. A root cause of this issue might be the low barriers of entry, as signing up to Google AdSense does not require any technical efforts, while also being free of charge (Google 2020). Consequently, even publishers with little website traffic are encouraged to sign up for the service. This results in a high number of domains, for whom the AdSense revenues stay at zero for a considerable timeframe. Different from missing observations, imputation is not a viable option for zero observations. This is due to the fact, that it is not possible to distinguish between actual zero observations, as there simply have not been any ad clicks and more technical causes like the discontinued usage of the AdSense network. Further, the main issue is not zero observations per se, but rather the missing variability over this time horizon, such that no trends or seasonalities can be observed, which makes it harder for any deep learning model to learn. Finally, it should be pointed out, that many small publishers also tend to inherit significantly more volatile revenue patterns than more established ones, which might hamper the transferability of the model's learnings across differently sized publishers. To combat the prevailing issues, two measures have been taken. First, similar to Rebortera and Fajardo (2019), all publishers with more than ten consecutive zero revenue observations have been removed and second, only publishers with more than five USD average daily revenues are retained. During preprocessing, the size of the dataset has been reduced tremendously, resulting in 42 domains remaining in the final dataset.

Even though no extensive exploratory data analysis will be discussed, one important finding needs to be highlighted. Based upon the autocorrelation plots in Appx. D, I find evidence for the hypothesis from Sec. 2.1, that web traffic is likely to have the most effect on short-term AdSense revenue patterns. In particular, the AdSense revenues showcase similar autocorrelation patterns like ad impressions (proxy for web traffic), only with less amplitude. CPC and CTR on the other hand do not inherit any noteworthy short-term autocorrelation patterns. It should be noted that the time series were aggregated across all publishers to produce insightful plots.

Similar to what Flunkert, Salinas, and Gasthaus (2017) find, another property of the data is that the distribution of the revenue time series across the different domains is strongly

skewed. Generally, this will make standardization techniques like normalization less effective. To conquer this problem, two measures have been taken. First, all daily AdSense revenues are log-transformed. Second, standard scaling is applied for each domain separately. Thus, all time-series are aggregated by domain and then each time series is individually normalized to zero mean and unit variance.

All deep learning models deployed in this study take as input a sequence of daily observations, following the sliding window technique, e.g. deployed by Vafaeipour et al. (2014) (illustration in Appx. C). Therefore, the inputs are two-dimensional matrices with the multivariate features on one axis and the input time steps on the other axis ($k_{max} = 89$). Hereby, k_{max} was chosen after inspection of the AdSense revenue autocorrelation plot (Appx. D). Similarly, with $\tau_{max} = 30$, the target is given by a vector of thirty univariate observations. To generate the next sample, both the input and output windows are shifted one day forward. In time series forecasting, train, validation, and test splits need to be generated carefully, to avoid any data leakage. Specifically, forecasts will usually become less accurate as predictions are made further in the future compared to the training time period. Thus, it is important to ensure that all training data stems from dates occurring before any date which is part of the validation and test sets. In this study, only observations before the 08.04.2020 are used for training, while data until the 19.06.2020 is used for validation and all remaining dates are used for testing.

It is important to notice, that the data lacks any information on the publishers' personal AdSense settings. This implies, that the positioning and the number of AdSense banners for any publisher are unknown. Consequently, even for companies in the dataset which are geographically and categorically close, the revenue transmitters can behave differently. Further, a publisher's settings might only allow for ads associated with certain keywords or ban specific keywords from their website. Analogously, the ban and the selection of keywords might result in domains with similar characteristics but different CTR and CPC ratios.

3.2 K-Means Clustering with Dynamic Time Warping

As elaborated in Sec. 1, one goal of this study is to test whether covariate time series, retrieved from peekd's database, can be fed as an input to the model and increase performance. In case

this claim holds true, it would yield an additional edge for the usage of peekd's proprietary database in the prediction of AdSense revenues.

To find covariate time series, at best, for every domain the competitors and/or substitutes could be identified and used as an input. Clearly, this would involve significant industry insight and manual work. Therefore, in line with Sec. 2.1, the most natural fit would be to feed the average time series of a domain's category or country as covariates to the model. Unfortunately, due to the rigorous data cleaning, the overlap between the domains, countries, and categories is limited. Hence, feeding time series from a domain's category or country does not provide a viable option, as often there would only be a single domain associated with a category/country. Alternatively, K-Means clustering with dynamic time warping (DTW) was deployed to aggregate meaningful groups of similar domains. This unsupervised algorithm aims to identify companies, which embed similar temporal patterns without providing more obvious categorical signals concerning their similarity. While the K-means clustering algorithm is well-known, the DTW offers a unique advantage for time series data. Compared to the Euclidian distance, which can only compare observations occurring at the same point in time, DTW can pick up similarities across shifted time series. DTW works by constructing *nxm* matrices for each pair of time series, where each element i, j in the matrix represents the Euclidean distance between two points in the pair of time series (Tavenard et al. 2020). Given the AdSense revenue time series from two domains $y_{1,t-k:t}$ and $y_{2,t-k:t}$, the objective is to find the shortest path through the matrix, which is the same as minimizing the cumulative distance. The objective can be denoted as:

$$DTW(y_{1,t-k:t}, y_{2,t-k:t}) = \min_{\pi} \sqrt{\sum_{(i,j)\in\pi} d(y_{1,i}, y_{2,j})} \tag{5}$$

where $\pi = [\pi_0, ..., \pi_K]$ and each element $\pi_k = (i_k, j_k)$ denotes an index pair within the distance matrix, condition to $\pi_0 = (0, 0)$, $\pi_k = (n-1, m-1)$. π_k is related to π_{k-1} by $i_{k-1} \leq i_k \leq i_{k-1}+1$ and $j_{k-1} \leq j_k \leq j_{k-1}+1$ (Tavenard et al. 2020). In the case at hand, the K-Means algorithm with DTW is deployed to find clusters using the AdSense revenue time series over the entire train period. Additionally, using DTW barycenter averaging, the algorithm produces centroids, which are time series in themselves (Petitjean, Ketterlin, and Gançarski 2011). Unfortunately, these centroids cannot be fed as covariates, as this would result in a significant data leakage

problem. In particular, at each time step, the value of the centroid time series is computed as the average of all its associated revenue time series, but in DTW space. As explained above, this might include shifted data points, and therefore also future values. Consequently, I use the clusters produced from K-Means with DTW as a static input. To produce covariate time series, I group all time-series based upon its cluster and compute the Euclidean barycenters at each time step, preventing any data leakage.

3.3 Quantile Loss

The objective function of any deep learning model translates a qualitative problem into a function to be minimized. In the case at hand, the Quantile Loss is used as the objective, such that AdSense revenue prediction is framed as a quantile regression problem. In the context of uncertainty and business planning, it is always desirable to not only generate a point prediction but also a corresponding measure of certainty. Quantile predictions over multi-step-horizons are two-dimensional, with predictions for every quantile at every time step in the prediction window. An individual quantile prediction always aims to overfit the actual value q% of the time. It should be noted that if $q = 0.5$, the quantile loss function reduces to the mean absolute error function, such that the median outcome is predicted, which will be treated as the point prediction. The quantile loss is given by:

$$\begin{aligned} L(\Omega, W) &= \sum_{y \in \Omega} \sum_{q \in Q} \sum_{\tau=1}^{\tau=max} \frac{QL(y_\tau, \hat{y}_\tau, q)}{M \tau_{max}} \\ QL(y, \hat{y}, q) &= max(q(y - \hat{y}), (1 - q)(\hat{y} - y)) \end{aligned} \tag{6}$$

where Ω is the training data with M samples (Lim et al. 2020) and the quantiles are set to $Q = \{0.25, 0.5, 0.75\}$. Thus, $q_{0.75} - q_{0.25}$ can be interpreted as 50% confidence interval.

3.4 Model selection

The model selection has been driven by the different forecasting strategies, reviewed in Sec. 2.2.2, and the search for a model that adequately handles the different input features available. Given the output-time horizon of $\tau_{max} = 30$, and the limited access to computational resources, no single-step models have been considered.

When recalling Eq. 1, it is important to note that all $x_{i,t}$ denote covariate time series, which are

known over the input horizon, but unknown over the output sequence (see also Appx. B). Most recent state of the art autoregressive models like Amazon's DeepAR (Flunkert, Salinas, and Gasthaus 2017) and transformer models, like the one by Li et al. (2019), ConvTrans hereafter, aim to model $\prod_{t=t_0+1}^{t=\tau} p(y_{i,t}|\ y_{i,t-1}, u_{i,t}; \Phi)$, where Φ denotes the learnable parameters and the complete problem is reduced to a one-step-ahead recursive problem. Importantly, $u_{i,t}$ continues to denote associated time-based covariates, which are known over the entire time period as in Eq. 1. In line with Sec. 2.2.2, neither of the two models mentioned above, but also no other recursive model can directly handle unknown covariates. Therefore, if one wants to lever these models for the problem at hand, either unknown covariates need to be dropped or forecasted separately.

For that reason, only models are deployed, that directly output multi-step predictions. Next to two baseline models, the recently released Temporal Fusion Transformer (TFT) by Lim et al. (2020) is implemented, which neatly aligns with the requirements of the prediction problem at hand. In particular, the authors intend to predict outputs for multiple entities *i* over a multi-step time-horizon, where an entity is a publisher in this study. Further, the model treats known and unknown time series features, as well as static variables separately (Lim et al. 2020). Another advantage stems from the option to interpret the predictions made by the TFT. The introduction of interpretable MSA layers and variable selection networks (VSN), allows for the analysis of temporal attention as well as feature importance (Lim et al. 2020). Ultimately, for multi-step time-horizon problems, the model was also shown to outperform autoregressive models like DeepAR and ConvTrans. The TFT architecture is explained in Sec. 3.5. For direct comparison a Seq2Seq LSTM (Sutskever, Vinyals, and Le 2014) with a Bahdanau attention layer (Bahdanau, Cho, and Bengio 2014) is implemented, sharing a lot of the underlying ideas with the TFT model, details are given in Sec. 3.6. As final baseline, a regular LSTM RNN directly outputting a multi-time-step vector is deployed (Hochreiter and Schmidhuber 1997).

3.5 Temporal Fusion Transformer

The TFT by Lim et al. (2020) is built from multiple processing layers stacked on top of each other (illustration in Appx. F). Before sequentially building up the TFT, I review the gating

mechanism used throughout the model.

Instead of regular activations, the TFT utilizes gates, which build on the two ideas of gated activations and residual connections, from Sec. 2.2.1. The intention is to provide the model with the freedom to decide on the amount of required non-linear processing for every input separately. When no further complexity is required, no non-linear transformation at all can be applied. Besides, a second optional input, the context vector can be supplied to the gate, e.g. allowing static inputs to influence the processing of time series features. With primary input a and optional context vector c, the GRN is computed as:

$$\begin{aligned} GRN_w(a, c) &= LayerNorm(a + GLU_w(\eta)) \\ \eta &= W_{1,w} ELU(W_{2,w} a + W_{3,w} c + b_{2,w}) + b_{1,w} \end{aligned} \tag{7}$$

where all $W_{(.)} \in \mathbb{R}^{d_{model} \times d_{model}}$ and $\eta(.) \in \mathbb{R}^{d_{model}}$ is an intermediate output, ELU is the exponential linear unit and d_{model} is the internal hidden size of the TFT and can be tuned. Layer Normalization is applied as in Ba, Kiros, and Hinton (2016). The addition of a in the first line reassembles Eq. 3, while the GLU is the same as in Eq. 2.

3.5.1 Variable Selection Layer

Before entering the actual model, encoder, decoder, and static inputs are preprocessed separately by a dedicated VSN. Importantly, variable selection takes place at each time step t separately. This processing step aims to increase interpretability through feature selection and filter out features that fail to contribute to model performance. Prior to the VSN application, all categorical and temporal features are embedded into d_{model} dimensional vectors using linear transformations and entity embeddings respectively. As an example the VSN for the decoder works as follows: Let $\xi_t^{(j)} \in \mathbb{R}^{d_{model}}$ denote the embedding for decoder variable j at time step t and $\Xi_t = [\xi_t^{(1)^T}, ..., \xi_t^{(m_x)^T}]^T$ the flattened concatenation of all decoder variable embeddings. The variable selection is performed by computing weights for each variable and applying them to the corresponding feature by simple multiplication, finally, all features are combined by summation. Thereby, weights are computed using a GRN_{v_χ} from Eq. 7. Static enrichment takes place by passing the static variable embeddings as context vector c_s through the GRN_{v_χ} (for the static

variable selection, there is no context vector). The VSN follows as:

$$\begin{aligned} v_{\chi_t} &= Softmax(GRN_{v_\chi}(\Xi_t, c_s)) \\ \tilde{\xi}_t^{(j)} &= GRN_{\tilde{\xi}(j)}(\xi_t^{(j)}) \\ \tilde{\xi}_t &= \sum_{j=1}^{m_x} v_{\chi_t^{(j)}} \tilde{\xi}_t^{(j)} \end{aligned} \tag{8}$$

where $v_{\chi_t} \in \mathbb{R}^{m_x}$ holds a weight for each variable. From Eq. 8, it can be seen that before the variable embeddings are scaled by the selection weights the embeddings are transformed through a dedicated $GRN_{\tilde{\xi}(j)}$ themselves, with weights shared across all time steps.

3.5.2 Locality Enhancement Layer

Locality enhancement is performed through a Seq2Seq LSTM. This is important, as the relevance of individual observations often only becomes visible within their context. A typical example is a change point, which can only be identified in context. The outputs of the local enhancement layer are denoted as $\phi(t, n)$ where $n \in [-k, t + \tau_{max}]$ is a positional index. The inputs for the encoder are $\tilde{\xi}_{t-k:t}$ and $\tilde{\xi}_{t:\tau_{max}}$ for the decoder. The processing from $\tilde{\xi}_{t-k:t}$ and $\tilde{\xi}_{t:\tau_{max}}$ to $\phi(t, n)$ is achieved by the application of the corresponding encoder/decoder LSTM cell. The LSTM's cell and hidden states are initialized using the outputs from two separate static input VSNs, once again allowing static inputs to influence the time series processing. Finally, another residual skip connection over the layer and another $GLU_{\tilde{\phi}}$ are applied:

$$\tilde{\phi}(t, n) = LayerNorm(\tilde{\xi}_{t+n} + GLU_{\tilde{\phi}}(\phi(t, n))) \tag{9}$$

3.5.3 Static Enrichment Layer

The adjacent layer is committed to static enrichment through the application of a GRN_Θ as in Eq. 7. Where $\tilde{\phi}(t, n)$ is transformed to $\Theta(t, n)$ by enrichment with c_e, the static context obtained through a VSN. The weights of the GRN_Θ are shared across all time steps.

3.5.4 Interpretable Multihead Attention Layer

Next, the interpretable MSA layer allows the model to pick up long-term trends, by accessing information from the far past. Specifically, all $\Theta(t, n)$ are concatenated as $\Theta(t) =$

$[\Theta(t,-k),...,\Theta(t,\tau)]^T$. Notably, the attention layer will apply forward masking, such that causal information flow is retained. Then, at each forecast step interpretable MSA is applied:

$$\begin{aligned} B(t) &= InterpretableMultiHead(\Theta(t),\Theta(t),\Theta(t)) \\ InterpretableMultiHead(Q,K,V) &= \tilde{H}W_H \\ \tilde{H} &= (1/H\sum_{h=1}^{m_H} A(QW_q^{(h)}, KW_k^{(h)}))VW_v \\ A(Q,K) &= Softmax(QK^T/\sqrt{d_{attn}}) \end{aligned} \tag{10}$$

where $B(t) = [\beta(t,-k),...,\beta(t,\tau)]$, $d_V = d_{attn} = d_{model}/m_H$ and the number of heads m_H, is a hyperparameter to tune. Each individual head reassembles Eq. 4, only that self-attention, as in any transformer architecture, uses $Q = K = V = \Theta(t)$, with $W_q^{(h)} \in \mathbb{R}^{d_{model}\times d_{attention}}$ and $W_k^{(q)} \in \mathbb{R}^{d_{model}\times d_{attention}}$ being learnable and specific to each head. And in the last line with $K \in \mathbb{R}^{N\times d_{attention}}$, $Q \in \mathbb{R}^{N\times d_{attention}}$, and $V \in \mathbb{R}^{N\times d_v}$. Different from the original MSA used in transformers (Vaswani et al. 2017), the TFT allows for interpretability. To do so, simply $W_v \in \mathbb{R}^{d_{model}\times d_v}$ needs to be shared across all heads. This allows for the computation of overall attention via summation over the heads, where each head can still learn different temporal patterns while attending the same set of inputs. $W_H \in \mathbb{R}^{d_{attention}\times d_{model}}$ is a simple linear transformation. Finally, $\beta(t,n)$ is transformed to $\delta(t,n)$ through application of another residual skip and GLU_δ unit as in Eq. 9, so that $\Theta(t,n)$ can skip the attention layer if necessary.

3.5.5 Positionwise Feed-Forward Layer

Lastly, $\delta(t,n)$ is transformed to $\psi(t,n)$ using a GRN_τ as in Eq. 7, with weights shared across all time steps. Finally, another residual skip and $GLU_{\tilde{\psi}}$ as in Eq. 9, transform $\psi(t,n)$ to $\tilde{\psi}(t,n)$, allowing the outputs from locality enhancement $\phi(t,n)$ to skip the entire transformer.

3.6 Seq2Seq LSTM RNN with Attention

The most natural choice for a benchmark to the TFT is a Seq2Seq model with a regular attention layer (illustration in Appx. G). Next to the VSN, the main contribution of the TFT is the transformer structure stacked on top of the locality enrichment layer (Sec. 3.5.2). Thus, the TFT should prove to outperform a regular attention layer stacked on top of a Seq2Seq LSTM model. In the encoder, the hidden states $h(t,n)$ with positional index $n \in [-k,t]$ are computed

by application of the respective LSTM cell, which takes in $h(t, n-1)$, as well as the concatenated inputs $x(t, n), u(t, n), y(t, n)$ (see Eq. 1). The hidden state of the last encoder cell is passed on to the decoder. Following Bahdanau, Cho, and Bengio (2014), an attention layer is deployed, where the context vectors are computed as $c(t, n)$ with positional index $n \in [t, t + \tau_{max}]$:

$$\begin{aligned} c(t,n) &= \sum_{i=t-k}^{T=t} \alpha(t,n)_i h(t)_i \\ \alpha(t,n)_i &= \frac{exp(e(t,n)_i)}{\sum_{j=t-k}^{T=t} exp(e(t,n)_j)} \\ e(t,n)_i &= V^T tanh(W[g(t,n-1), h(t)_i]) \end{aligned} \tag{11}$$

where $V \in \mathbb{R}^{d_{hidden\,enc} \times 1}$ and $W \in \mathbb{R}^{d_{hidden\,enc} \times (d_{hidden\,dec} + d_{hidden\,enc})}$ are learnable, $g(t, n-1)$ is the hidden state of the previous decoder cell, $h(t)_i$ is the hidden encoder state at encoder step i and $\alpha(t, n)_i$ is the attention weight used at forecasting step n for encoder step i. Eq. 11, can be mapped to the to the general form of attention (Eq. 4), as explained in Sec. 2.2.1. Different from the TFT, the decoder of the Seq2Seq LSTM works recursively. To make a prediction $\hat{y}(t, n)$, the prediction from the previous time step $\hat{y}(t, n-1)$ is concatenated to the decoder inputs, which are the context vector $c(t, n)$, known future inputs $u(t, n)$ and the static inputs s, which are repeated for each decoder cell. Finally, the decoder hidden states $r(t, n)$ with positional index $n \in [t, t + \tau_{max}]$ are computed by application of the respective LSTM cell.

4 Results & Interpretation

4.1 Hyperparameter Tuning

Hyperparameter tuning was conducted using the Bayesian optimization framework Optuna, where due to resource constraints, each model was given 20 trials with five epochs each (Akiba et al. 2019). Even though the number of epochs might appear small, each epoch works on all samples generated using a one-step forward sliding window. As such, there will be significant overlap of the daily observations visited across samples. Therefore, the models will converge already after a very low number of epochs. For the TFT, hidden size, dropout, and number of heads were tuned. In the Seq2Seq architecture, hidden sizes in encoder and decoder, respective dropout rates, and the number of layers in the encoder were tuned. Finally, in the LSTM, hidden

layer size, number of layers, and dropout were optimized. The batch size was set to 32 and the Adam optimizer was used for all models. The learning rate was only tuned from a subset of 0.001 and 0.0001, all in an effort to reduce hyperparameter space (results in Appx. H).

4.2 Results

As expected, the TFT outperforms the other models across all metrics and experiment settings. Nevertheless, the performance improvements are moderate. In particular, considering the case without peekd insight, the TFT offers a 8.9%/11.6% improvement over the LSTM and a 6.9%/8.8% improvement over Seq2Seq, with respect to the quantile and symmetric mean absolute percentage error loss (SMAPE, see Appx. E). In table 1, the two columns without and with peekd insight, distinguish the case where the Euclidean barycenter time series was fed as an additional covariate and the cluster (see Sec. 3.2) as an additional static variable from the case where only the remaining features were used (Appx. B). I find that the peekd insight has not helped to improve the model.

For the unsupervised K-Means clustering with DTW, the number of clusters was set to three based on an elbow analysis (see Appx. I). Cluster zero is assigned 13 domains, cluster one 24 domains, and cluster two only five domains. The Euclidean barycenters can be found in Appx. I. Further, also value counts for domains per category, country, and cluster can be found in Appx. I. In line with Sec. 3.2, it does not appear that the DTW clusters can be reconstructed using simple geographical or categorical groupings, thus they might indeed embed temporal similarities. Nevertheless, already the unbalanced assignment provides a possible explanation for the unaltered model performance. Clusters, which are too large might not provide insights by smoothing out temporal patterns, while very small clusters might embed too much noise of individual publishers. Notably, if the cluster assignment would group e.g. competitors, a small cluster size would be beneficial. As explained earlier, the dataset does not allow for such grouping (see Sec. 3.2). Second, it seems like even though DTW was deployed, the main difference between the Euclidean barycenters stems from the different levels of AdSense revenues, not similar temporal patterns. This becomes especially clear when comparing the barycenters of clusters one and two (Appx. I). To some extent, both seem to embed similar

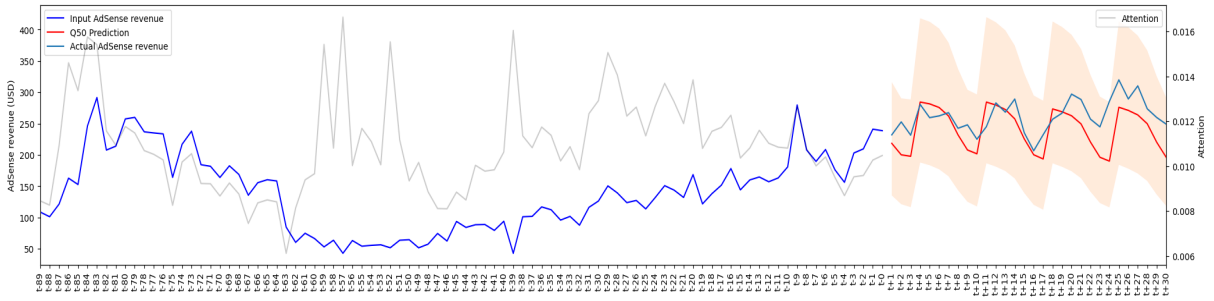

Figure 1: TFT without peekd insight - Top 1 prediction

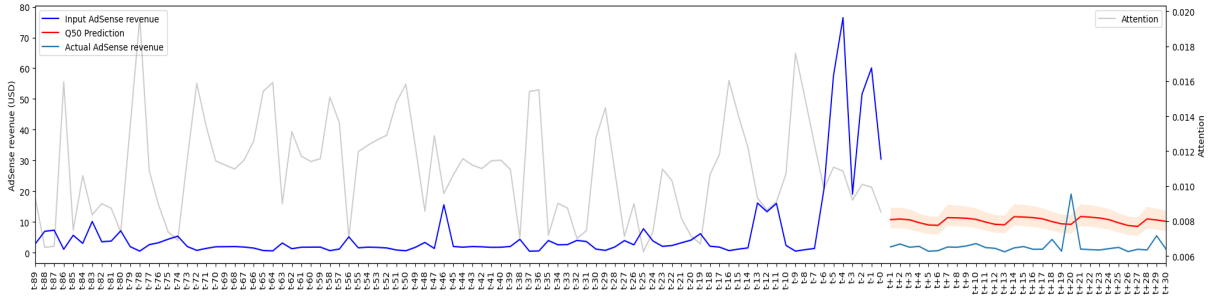

Figure 2: TFT without peekd insight - Worst 1 prediction

patterns but only at different levels. Certainly, both these observations hint at why the clustering failed to improve predictive power.

Model	w/o peekd insight		w/ peekd insight	
	Quantile Loss	SMAPE	Quantile Loss	SMAPE
TFT	0.832	0.831	0.833	0.826
Seq2Seq	0.894	0.911	0.936	0.979
LSTM	0.913	0.940	0.917	0.945

Table 1: Study Results

It is worth taking a look at the generated predictions. The best and worst TFT predictions without access to additional peekd features are presented in Fig. 1 and Fig. 2. On inspection of the best observation, it becomes clear that the TFT model picks up the weekly patterns very well. At forecast step $t + 2$, the first turning point occurs, which is afterwards repeated every seven days. Nevertheless, there also seems to be an upward trend present from $t - 60$ onwards. Predictions are getting worse as time passes and the level of the predictions remains constant. Looking at the worst prediction provides an indication of the reason, even sufficiently complex models like the TFT seem to occasionally struggle to compute sensible predictions. While revenues remain flat until $t - 6$, there is a significant revenue peak right before the prediction period starts. The model subsequently picks a wrong revenue level throughout the forecasting period. As elaborated in section 2.1, an explanation for the unexpected peaks might be the untracked advertisers' actions.

4.3 Interpretation

In this section, the results from the TFT without additional peekd features will be levered to generate insights into the underlying dynamics of the AdSense revenue predictions.

Feature importance is analyzed by interpreting the variable selection weights used by the TFT. As described in Sec. 3.5.1, the TFT has been producing sparse weights for static, encoder, and decoder inputs at the corresponding time steps. With the aim to interpret the weight matrices across all time steps, mean reduction across time has been deployed. Unsurprisingly, the encoder puts most of its weight on the AdSense revenue (Fig. 3). Additionally, ad impressions, the day of the week, and page views dominate the encoder variable selection. This observation can be interpreted as the first piece of evidence for what has been proposed on the basis of existing literature (Sec. 2.1) and the analysis of the autocorrelation plots (Appx. D), that within a short-term prediction horizon, the web traffic might yield the most significant determinant of AdSense revenues. As explained earlier, ad impressions are a suitable dummy for web traffic, while the day of the week, is a temporal variable, which has previously been found to impact web traffic (Li and Moore 2008). Nevertheless, these findings do not oppose the results from previous literature (Sec. 2.1), indicating that there are three main transmitters of AdSense revenues, namely web traffic, CTR, and CPC. Indeed, it should be noted that given the past ad impressions, ad clicks, and AdSense revenues, the model can easily compute CTR and CPC as linear combinations. But given the assumption that both CTR and CPC stay rather constant over the forecast period, they do not add a lot of information to the revenue development as the model also has direct access to past AdSense revenues. Similarly, the decoder puts the most weight on the day of the week (Fig. 4), supporting the findings from the encoder VSNs. Finally, the static variable importance is dominated by the category (Fig. 5). As explained in Sec. 3.1, the dataset was already filtered by country, which might explain its low feature importance. The category, on the other hand, might provide a tangible grouping of publishers with similar transmitters and temporal patterns. Further, the lowest importance is attributed to the categorical id, which uniquely identifies a single publisher. This is a crucial observation as it indicates, that the model indeed has not tried to mimic single models learned for each cross-sectional instance independently. Rather, in line with Flunkert, Salinas, and Gasthaus (2017), the model learns

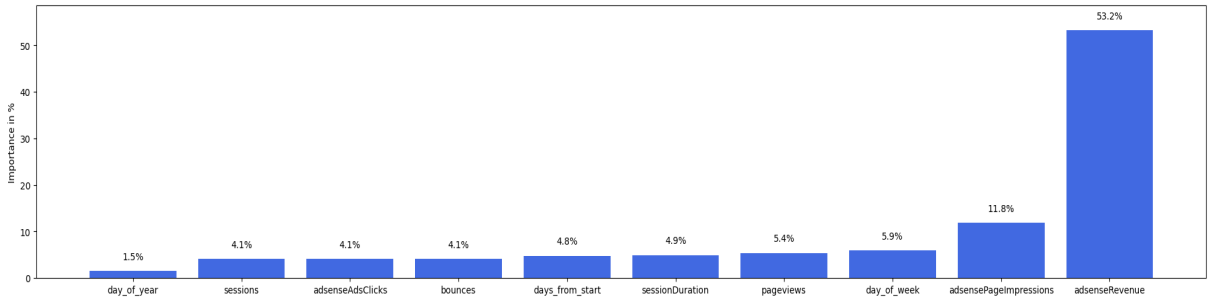

Figure 3: Encoder feature importance

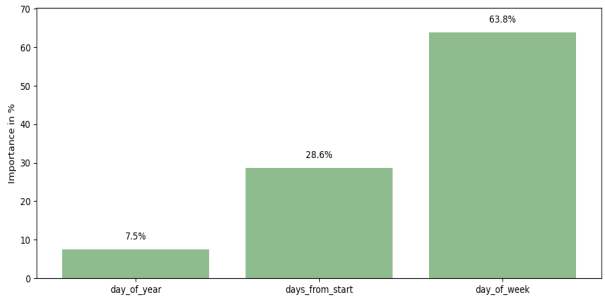

Figure 4: Decoder feature importance

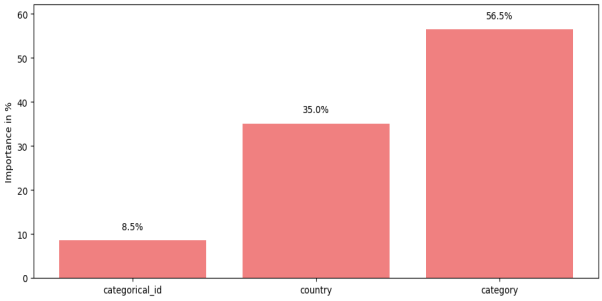

Figure 5: Static feature importance

transferable feature representations and thus benefits from the access to cross-sectional data. In addition, also the temporal attention of the TFT can be interpreted. As explained in Sec. 3.5.4, all attention heads attend the same values and overall attention is computed by additive aggregation of the individual heads (Lim et al. 2020). The MSA layer will accordingly generate a time to attend by time step map. The focus in this analysis has been put on the attention of the decoder steps over the encoder steps. In particular, the attention weights from the first decoder step are used for the purpose of interpretation. Due to masking, it is not possible to perform any reduction operation across the decoder steps as each decoder step has access to an input sequence of different length. In line with the individual predictions presented in Sec. 1, the TFT is mainly concerned with the weekly recurring patterns inherent in the data, which once again, can be treated as evidence for the weekly web traffic dominating AdSense revenue development in the short-term (Fig. 4).

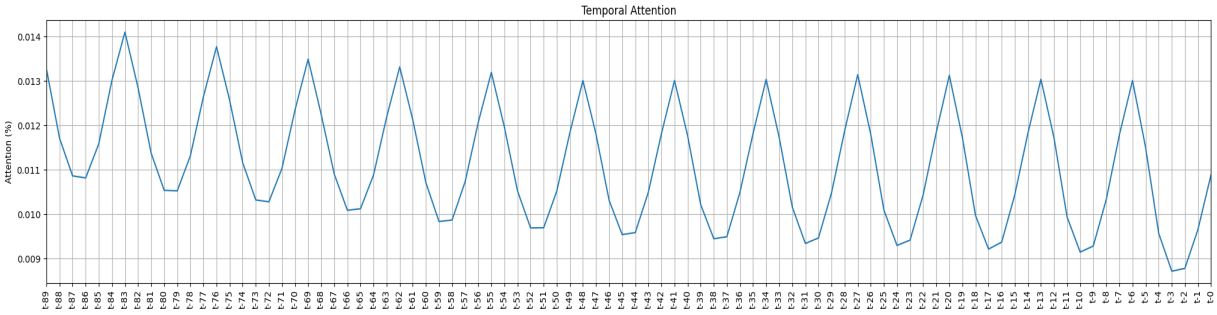

Figure 6: Attention over input sequence

5 Conclusion & Limitations

It was successfully shown, that AdSense revenues are predictable and embed temporal patterns. The results from this study shall act as a case study, pointing towards the predictability of AdSense revenues from peekd's database. As peekd acquires more customers and consequently access to more AdSense revenue data, the model performance is expected to further improve. Even though based on the sample levered in this study, feeding additional covariates created from the database has not improved model performance, the argument in favor of such methodology in a larger scale study remains, as more sensible groupings could be built. Nonetheless, the model developed in this study confidently offers a competitive edge for peekd. First, no competing products exist. Second, the importance of the categorical id in generating predictions indicates that also in the case at hand, sufficiently complex deep learning models trained on cross-sectional time-series outperform models learned from a single instance dataset (Flunkert, Salinas, and Gasthaus 2017).

With respect to the existing body of research on Google AdSense, I find indications that for publishers' past AdSense revenues, the time series itself, ad impressions, the day of the week, and page views are most important when making predictions. I find that the TFT mainly learns weekly temporal attention patterns, which are usually embedded in web traffic data. This might act as an indication, that while CTR and CPC seem to be less variable in the short-term, website traffic patterns directly transfer into short-term movements of AdSense revenues. Finally, evidence is collected, that publishers from similar categories tend to embed similar temporal patterns, possibly as they remain associated with similar keywords.

Concerning deep learning applications in a business setting, this study adds to the body of successful time series applications. The TFT, given its VSNs and interpretable MSA, offers a promising example of deep learning not only improving performance but also interpretability.

A final concern stems from the fact that peekd's most sought-after product is a keyword search algorithm, providing customers with information on its non-sponsored search queries. This product mainly attracts e-commerce shops, but far less so traditional news and media publishers. Thus, e-commerce shops might be over-represented in this study, for whom advertising is a secondary source of income.

References

Akiba, Takuya, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. "Optuna: A Next-generation Hyperparameter Optimization Framework". In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. "Layer normalization". *arXiv preprint arXiv:1607.06450.*

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. 2014. "Neural machine translation by jointly learning to align and translate". *arXiv preprint arXiv:1409.0473.*

Borovykh, Anastasia, Sander Bohte, and Cornelis W Oosterlee. 2017. "Conditional time series forecasting with convolutional neural networks". *arXiv preprint arXiv:1703.04691.*

Cerqueira, Vitor, Luis Torgo, and Carlos Soares. 2019. "Machine learning vs statistical methods for time series forecasting: Size matters". *arXiv preprint arXiv:1909.13316.*

Dauphin, Yann N, Angela Fan, Michael Auli, and David Grangier. 2017. "Language modeling with gated convolutional networks". In *International conference on machine learning,* 933–941. PMLR.

Edlich, Alex, Greg Phalin, Rahil Jogani, and Sanjay Kaniyar. 2019. *Driving impact at scale from automation and AI,* February. https://www.mckinsey.com/~/media/McKinsey/Business%20Functions/McKinsey%20Digital/Our%20Insights/Driving%20impact%20at%20scale%20from%20automation%20and%20AI/Driving-impact-at-scale-from-automation-and-AI.ashx.

Flunkert, Valentin, David Salinas, and Jan Gasthaus. 2017. "DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks". *CoRR* abs/1704.04110. arXiv: 1704.04110. http://arxiv.org/abs/1704.04110.

Google. 2020. *How AdSense works.* https://support.google.com/adsense/answer/6242051?hl=en.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. "Deep residual learning for image recognition". In *Proceedings of the IEEE conference on computer vision and pattern recognition,* 770–778.

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. "Long short-term memory". *Neural computation* 9 (8): 1735–1780.

Li, Jia, and Andrew W Moore. 2008. "Forecasting web page views: methods and observations". *Journal of Machine Learning Research* 9 (Oct): 2217–2250.

Li, Shiyang, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. 2019. "Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting". *CoRR* abs/1907.00235. arXiv: 1907.00235. http://arxiv.org/abs/1907.00235.

Liang, Jingxi. 2017. *Luong attention and Bahdanau attention,* August. http://cnyah.com/2017/08/01/attention-variants/.

Lim, Bryan, Sercan O Arik, Nicolas Loeff, and Tomas Pfister. 2020. "Temporal fusion transformers for interpretable multi-horizon time series forecasting". *arXiv preprint arXiv:1912.09363.*

Lim, Bryan, and Stefan Zohren. 2020. "Time Series Forecasting With Deep Learning: A Survey". *arXiv preprint arXiv:2004.13408.*

McMahan, H. Brendan, Gary Holt, D. Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, et al. 2013. "Ad Click Prediction: a View from the Trenches". In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD).*

Mozaffari, Ladan, Ahmad Mozaffari, and Nasser Azad. 2014. "Vehicle speed prediction via a sliding-window time series analysis and an evolutionary least learning machine: A case study on San Francisco urban roads". *Engineering Science and Technology, an International Journal* 6 (December). https://doi.org/10.1016/j.jestch.2014.11.002.

Oord, Aäron van den, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. 2016. "WaveNet: A Generative Model for Raw Audio". *CoRR* abs/1609.03499. arXiv: 1609.03499. http://arxiv.org/abs/1609.03499.

Petitjean, François, Alain Ketterlin, and Pierre Gançarski. 2011. "A global averaging method for dynamic time warping, with applications to clustering". *Pattern Recognition* 44 (3): 678–693.

Rebortera, Mariannie A, and Arnel C Fajardo. 2019. "An enhanced deep learning approach in forecasting banana harvest yields". *Editorial Preface From the Desk of Managing Editor* 10 (9).

Rumelhart, D., Geoffrey E. Hinton, and R. J. Williams. 1986. "Learning representations by back-propagating errors". *Nature* 323:533–536.

Sutskever, Ilya, Oriol Vinyals, and Quoc V Le. 2014. "Sequence to sequence learning with neural networks". *Advances in neural information processing systems* 27:3104–3112.

Tavenard, Romain, Johann Faouzi, Gilles Vandewiele, Felix Divo, Guillaume Androz, Chester Holtz, Marie Payne, Roman Yurchak, Marc Rußwurm, Kushal Kolar, et al. 2020. "Tslearn, a machine learning toolkit for time series data". *Journal of Machine Learning Research* 21 (118): 1–6.

Vafaeipour, Majid, Omid Rahbari, Marc A Rosen, Farivar Fazelpour, and Pooyandeh Ansarirad. 2014. "Application of sliding window technique for prediction of wind velocity time series". *International Journal of Energy and Environmental Engineering* 5 (2-3): 105.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. "Attention is all you need". In *Advances in neural information processing systems,* 5998–6008.

Wang, Chieh-Jen, and Hsin-Hsi Chen. 2012. "Learning to predict the cost-per-click for your ad words", 2291–2294. October. https://doi.org/10.1145/2396761.2398623.

Yu, Fisher, and Vladlen Koltun. 2015. "Multi-scale context aggregation by dilated convolutions". *arXiv preprint arXiv:1511.07122.*

Zhou, Guorui, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. "Deep interest network for click-through rate prediction". In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining,* 1059–1068.

A Country Filter

number	Country
0	Australia
1	Belgium
2	Canada
3	Denmark
4	Germany
5	Italy
6	Japan
7	Netherlands
8	Norway
9	Portugal
10	Spain
11	Sweden
11	Switzerland
12	United Kingdom of Great Britain and Northern Ireland
13	United States of America
14	category

Table 2: Country Filter

B Data Dictionary

number	Feature	Data Type	peekd Insight	Kown/Unknown	Description
0	adsenseRevenue	RealValue	False	Target	Daily online advertisement revenues
1	dayOfYear	RealValue	False	Known	-
2	dayOfWeek	RealValue	False	Known	-
3	dayFromStart	RealValue	False	Known	Days difference from the first observation
4	adsenseAdsClicks	RealValue	False	Unknown	Daily ad clicks
5	adsensePageImpressions	RealValue	False	Unknown	Daily ad impressions
6	pageviews	RealValue	False	Unknown	Daily page views
7	sessions	RealValue	False	Unknown	Daily sessions, a session is a users interaction with a website (can encompass multiple page views)
8	bounces	RealValue	False	Unknown	Daily bounces, number of users who only visit a single page and immediately leave
9	sessionDuration	RealValue	False	Unknown	Daily session duration, time users actively interact with a website
10	clusterAdsenseRevenue	RealValue	True	Unknown	Euclidean barycenter AdSense revenue according to domain's cluster assignment
11	categoricalId	Categorical	False	Static	Unique identifier for every domain
12	country	Categorical	False	Static	Origin of the domain
13	category	Categorical	False	Static	Category of the domain, assigned based on content
14	cluster	Categorical	True	Static	K-Means cluster assignment generated with DTW warping

Table 3: Data Dictionary

C Sliding Window Illustration

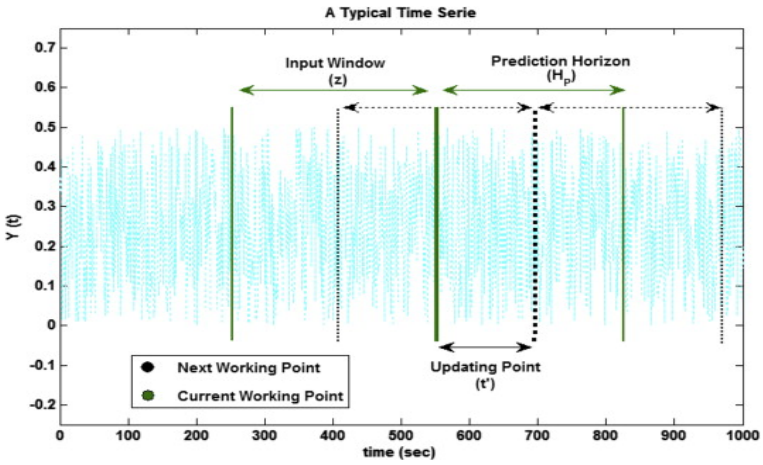

Figure 7: Sliding Window Illustration, from Mozaffari, Mozaffari, and Azad (2014).

D Autocorrelation Plots

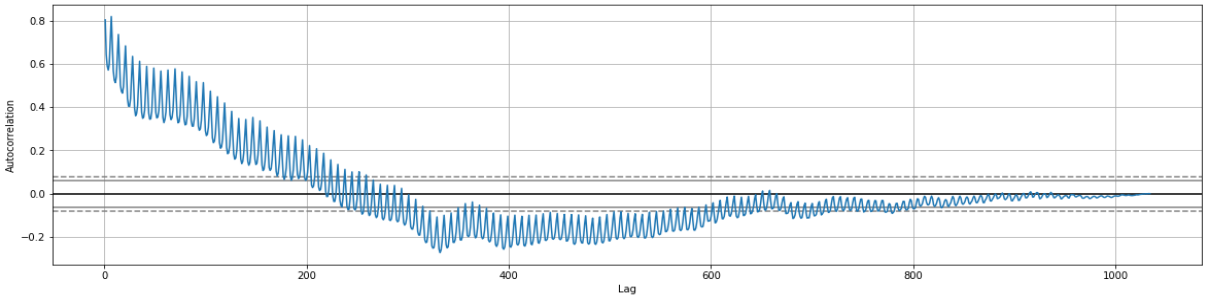

Figure 8: AdSenseRevenue Autocorrelation Plot

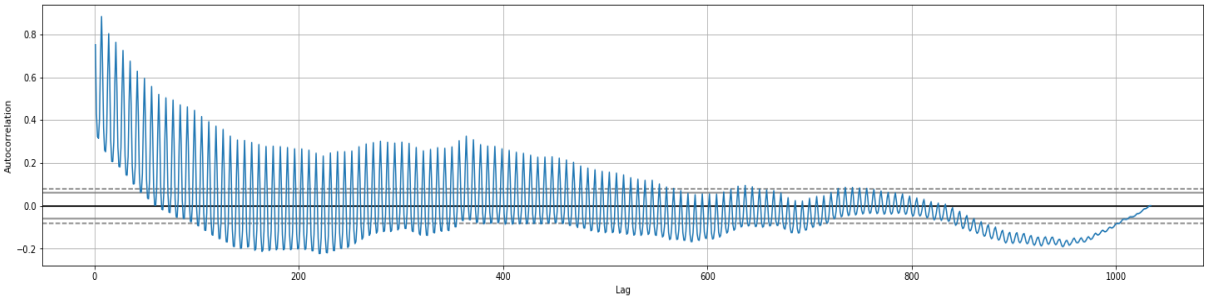

Figure 9: Page Impressions Autocorrelation Plot

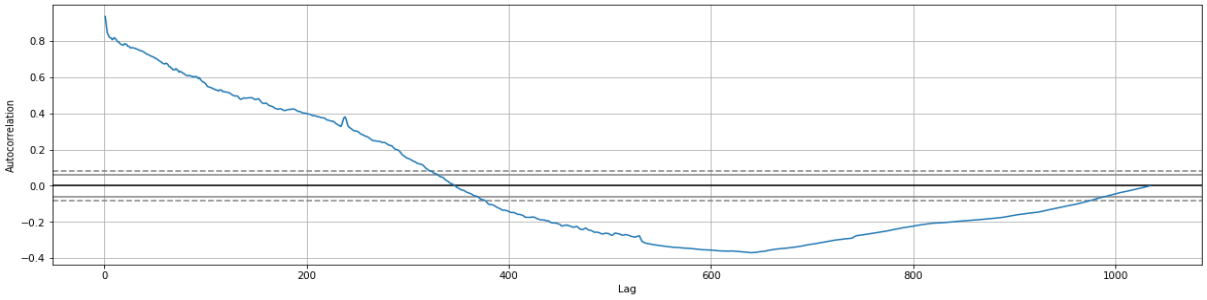

Figure 10: Click-Through-Rate Autocorrelation Plot

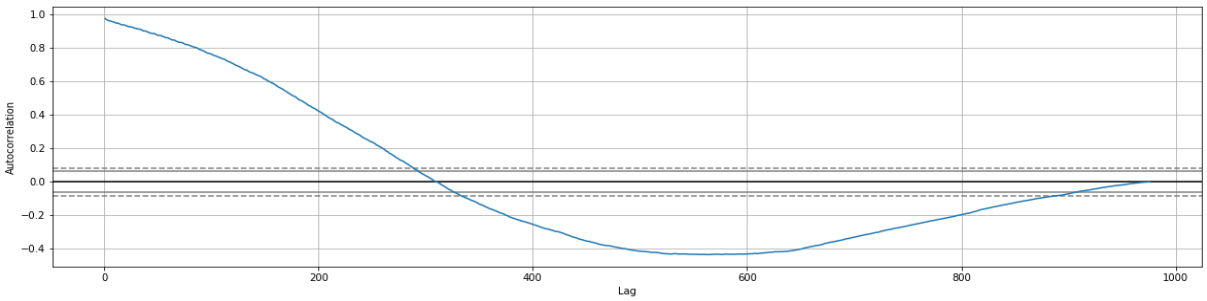

Figure 11: Cost-Per-Click Autocorrelation Plot

E Symmetric Mean Absolute Percentage Error

$$SMAPE(\Omega, W) = \sum_{y \in \Omega} \frac{1}{\tau_{max}} \sum_{\tau=1}^{\tau_{max}} \frac{|y_\tau - \hat{y}_\tau|}{(|y_\tau| + |\hat{y}_\tau|)/2} \tag{12}$$

where Ω is the training data with M samples.

F TFT Illustration

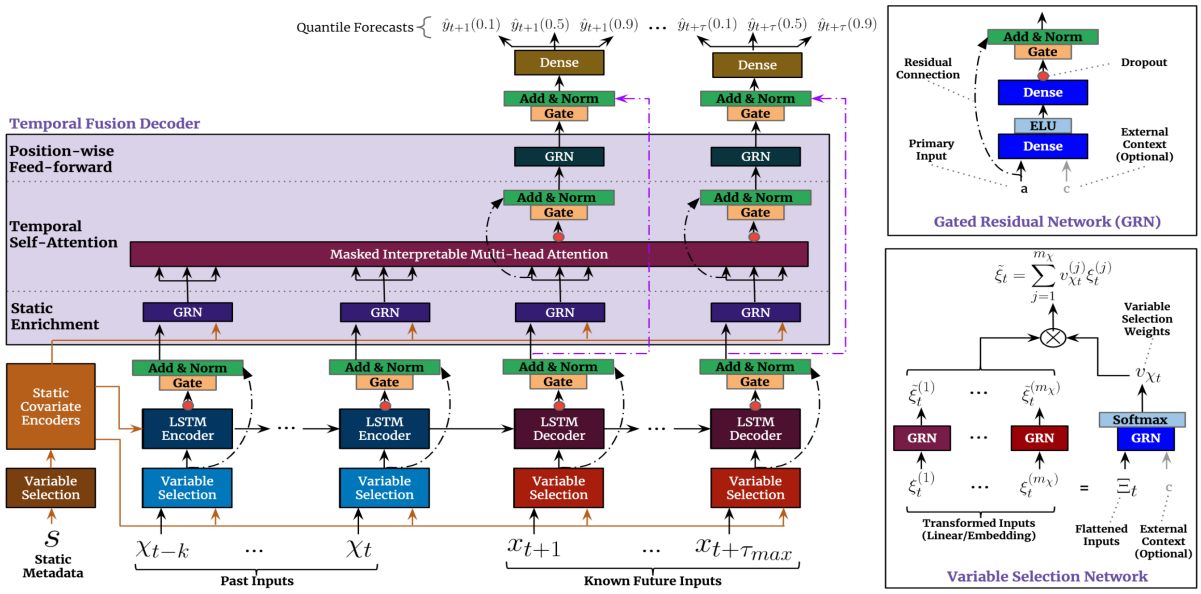

Figure 12: TFT model architecture, from Lim et al. (2020).

G Seq2Seq with Bahdanau Attention Illustration

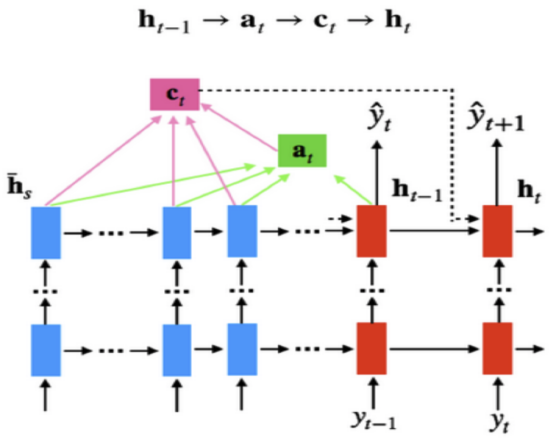

Figure 13: Seq2Seq with Bahdanau Attention, from Liang (2017).

H Hyperparameter Tuning

H.1 TFT Hyperparameter Tuning - without peekd insight

number	value	params_dropout_rate	params_hidden_layer_size	params_lr	params_num_heads	state
0	0.896935	0.4	64	0.0001	4	COMPLETE
1	0.834544	0.4	32	0.0001	1	COMPLETE
2	1.349725	0.8	16	0.0001	4	COMPLETE
3	0.860030	0.8	64	0.0001	4	COMPLETE
4	0.881148	0.6	16	0.0001	1	COMPLETE
5	0.815677	0.6	32	0.0001	4	COMPLETE
6	0.865248	0.4	32	0.0001	1	COMPLETE
7	1.342736	0.8	16	0.0001	1	PRUNED
8	1.331752	0.8	32	0.0001	4	PRUNED
9	1.331095	0.8	16	0.0001	4	PRUNED
10	0.840454	0.6	32	0.0001	4	PRUNED
11	0.813602	0.4	32	0.0001	1	COMPLETE
12	0.837854	0.6	32	0.0001	1	PRUNED
13	0.844723	0.6	32	0.0001	1	PRUNED
14	0.873541	0.4	32	0.0001	4	PRUNED
15	0.836640	0.6	32	0.0001	1	PRUNED
16	0.874717	0.4	64	0.0001	4	PRUNED
17	1.181574	0.6	32	0.0001	1	PRUNED
18	0.861368	0.4	32	0.0001	4	PRUNED
19	1.186880	0.6	32	0.0001	1	PRUNED

Table 4: TFT Hyperparameter Tuning - without peekd insight, only showing trials with learning rate 0.0001

H.2 TFT Hyperparameter Tuning - with peekd insight

number	value	params_dropout_rate	params_hidden_layer_size	params_lr	params_num_heads	state
0	0.847525	0.4	32	0.0001	4	COMPLETE
1	0.916774	0.6	16	0.0001	4	COMPLETE
2	0.953776	0.4	64	0.0001	1	COMPLETE
3	0.986895	0.4	16	0.0001	1	COMPLETE
4	0.840813	0.8	64	0.0001	4	COMPLETE
5	0.887442	0.4	32	0.0001	1	COMPLETE
6	1.336842	0.6	16	0.0001	1	PRUNED
7	1.357245	0.8	16	0.0001	1	PRUNED
8	0.906609	0.4	32	0.0001	1	PRUNED
9	0.884515	0.6	32	0.0001	4	PRUNED
10	1.290845	0.8	64	0.0001	4	PRUNED
11	1.373095	0.8	64	0.0001	4	PRUNED
12	1.356763	0.8	64	0.0001	4	PRUNED
13	0.847383	0.6	32	0.0001	4	COMPLETE
14	0.821525	0.6	32	0.0001	4	COMPLETE
15	1.325336	0.8	64	0.0001	4	PRUNED
16	1.257395	0.6	32	0.0001	4	PRUNED
17	1.363408	0.8	64	0.0001	4	PRUNED
18	1.086708	0.6	32	0.0001	4	PRUNED
19	1.383189	0.8	64	0.0001	4	PRUNED

Table 5: TFT Hyperparameter Tuning - with peekd insight, only showing trials with learning rate 0.0001

H.3 Seq2Seq Hyperparameter Tuning - without peekd insight

number	value	params_dec_D_hidden	params_dec_time_dropout	params_enc_D_hidden	params_enc_n_layers	params_enc_rnn_dropout	params_lr	state
0	0.906107	64	0.8	64	2	0.4	0.0001	COMPLETE
1	0.993403	64	0.6	64	1	0.4	0.0001	COMPLETE
2	1.029127	32	0.4	128	1	0.8	0.0001	COMPLETE
3	1.007250	32	0.8	128	1	0.8	0.0001	COMPLETE
4	0.989132	32	0.4	64	1	0.8	0.0001	COMPLETE
5	1.103307	32	0.8	128	1	0.8	0.0001	PRUNED
6	0.952987	64	0.4	128	2	0.6	0.0001	COMPLETE
7	1.101234	64	0.4	64	1	0.4	0.0001	PRUNED
8	1.075791	64	0.8	64	2	0.4	0.0001	PRUNED
9	0.972416	64	0.4	64	2	0.8	0.0001	PRUNED
10	0.972254	64	0.6	64	2	0.6	0.0001	PRUNED
11	0.968675	64	0.6	128	2	0.6	0.0001	COMPLETE
12	1.011685	64	0.8	128	2	0.6	0.0001	COMPLETE
13	1.008139	64	0.6	128	2	0.6	0.0001	COMPLETE
14	0.956872	64	0.6	128	2	0.4	0.0001	PRUNED
15	1.056037	64	0.4	64	2	0.6	0.0001	PRUNED
16	1.066990	64	0.8	64	2	0.4	0.0001	PRUNED
17	0.939042	64	0.6	128	2	0.6	0.0001	PRUNED
18	1.095852	64	0.4	64	2	0.4	0.0001	PRUNED
19	0.936970	64	0.8	128	2	0.6	0.0001	COMPLETE

Table 6: Seq2Seq Hyperparameter Tuning - without peekd insight, only showing trials with learning rate 0.0001

H.4 Seq2Seq Hyperparameter Tuning - with peekd insight

number	value	params_dec_D_hidden	params_dec_time_dropout	params_enc_D_hidden	params_enc_n_layers	params_enc_rnn_dropout	params_lr	state
0	0.959140	32	0.4	64	1	0.6	0.0001	COMPLETE
1	0.963109	32	0.4	64	1	0.4	0.0001	COMPLETE
2	1.018803	32	0.6	64	2	0.4	0.0001	COMPLETE
3	0.979243	64	0.4	128	1	0.8	0.0001	COMPLETE
4	0.987392	64	0.6	64	1	0.4	0.0001	COMPLETE
5	0.990670	64	0.4	64	1	0.6	0.0001	PRUNED
6	1.184946	32	0.4	64	1	0.6	0.0001	PRUNED
7	0.986777	64	0.8	64	1	0.6	0.0001	COMPLETE
8	0.988331	64	0.8	128	1	0.6	0.0001	COMPLETE
9	0.927765	64	0.4	64	2	0.6	0.0001	COMPLETE
10	0.972695	64	0.6	128	2	0.8	0.0001	PRUNED
11	0.941205	32	0.4	64	2	0.8	0.0001	COMPLETE
12	1.123837	32	0.4	64	2	0.8	0.0001	PRUNED
13	1.182609	32	0.6	64	2	0.8	0.0001	PRUNED
14	0.922154	64	0.4	64	2	0.8	0.0001	COMPLETE
15	1.099955	64	0.6	64	2	0.8	0.0001	PRUNED
16	1.022369	64	0.4	128	2	0.4	0.0001	COMPLETE
17	1.045986	64	0.4	64	2	0.6	0.0001	PRUNED
18	1.068359	64	0.6	64	2	0.8	0.0001	PRUNED
19	0.930426	64	0.8	64	2	0.6	0.0001	COMPLETE

Table 7: Seq2Seq Hyperparameter Tuning - with peekd insight, only showing trials with learning rate 0.0001

H.5 LSTM Hyperparameter Tuning - without peekd insight

number	value	params_D_hidden	params_dropout	params_lr	params_n_layers	state
0	0.942215	64	0.4	0.0001	1	COMPLETE
1	0.945441	64	0.6	0.0001	1	COMPLETE
2	0.926925	64	0.6	0.0001	1	COMPLETE
3	1.032526	64	0.4	0.0001	2	COMPLETE
4	0.918174	64	0.6	0.0001	1	COMPLETE
5	0.954240	128	0.6	0.0001	1	COMPLETE
6	0.925316	64	0.6	0.0001	1	PRUNED
7	0.968838	64	0.8	0.0001	2	PRUNED
8	1.005391	128	0.4	0.0001	1	COMPLETE
9	0.939982	64	0.8	0.0001	1	COMPLETE
10	1.026076	128	0.8	0.0001	2	COMPLETE
11	0.950871	64	0.6	0.0001	1	PRUNED
12	0.962231	64	0.6	0.0001	1	PRUNED
13	0.934528	64	0.6	0.0001	1	COMPLETE
14	0.945280	64	0.8	0.0001	1	PRUNED
15	0.915276	64	0.4	0.0001	1	COMPLETE
16	0.954010	64	0.4	0.0001	1	PRUNED
17	0.963135	128	0.4	0.0001	2	COMPLETE
18	0.969474	64	0.4	0.0001	1	PRUNED
19	0.978058	64	0.4	0.0001	1	PRUNED

Table 8: LSTM Hyperparameter Tuning - without peekd insight, only showing trials with learning rate 0.0001

H.6 LSTM Hyperparameter Tuning - with peekd insight

number	value	params_D_hidden	params_dropout	params_lr	params_n_layers	state
0	0.944035	64	0.4	0.0001	2	COMPLETE
1	0.908138	64	0.6	0.0001	1	COMPLETE
2	0.949033	128	0.6	0.0001	1	COMPLETE
3	0.887798	64	0.8	0.0001	1	COMPLETE
4	1.090801	128	0.4	0.0001	2	COMPLETE
5	0.994987	64	0.6	0.0001	1	PRUNED
6	0.959028	64	0.8	0.0001	2	PRUNED
7	0.952565	64	0.8	0.0001	2	PRUNED
8	0.917928	128	0.4	0.0001	1	PRUNED
9	0.933441	64	0.6	0.0001	2	PRUNED
10	0.992159	64	0.8	0.0001	1	PRUNED
11	0.989768	64	0.8	0.0001	1	PRUNED
12	0.959660	64	0.6	0.0001	1	PRUNED
13	0.958878	64	0.8	0.0001	1	PRUNED
14	0.963202	64	0.6	0.0001	1	PRUNED
15	0.999092	64	0.8	0.0001	1	PRUNED
16	0.964455	64	0.4	0.0001	1	PRUNED
17	0.903410	128	0.6	0.0001	1	PRUNED
18	0.939278	64	0.8	0.0001	1	PRUNED
19	0.983134	64	0.6	0.0001	1	PRUNED

Table 9: LSTM Hyperparameter Tuning - with peekd insight, only showing trials with learning rate 0.0001

I K-Means Clustering with DTW

I.1 Elbow Analysis

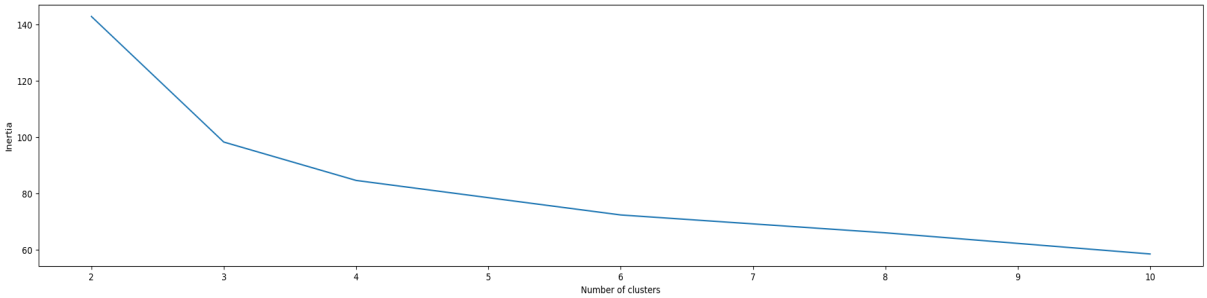

Figure 14: K-Means clustering with DTW - Elbow analysis

I.2 Euclidean barycenters

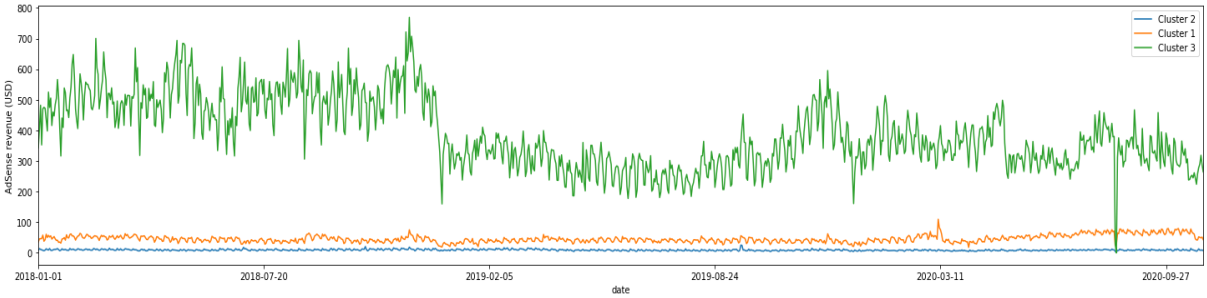

Figure 15: K-Means clustering with DTW - Euclidean barycenter AdSense revenue time series

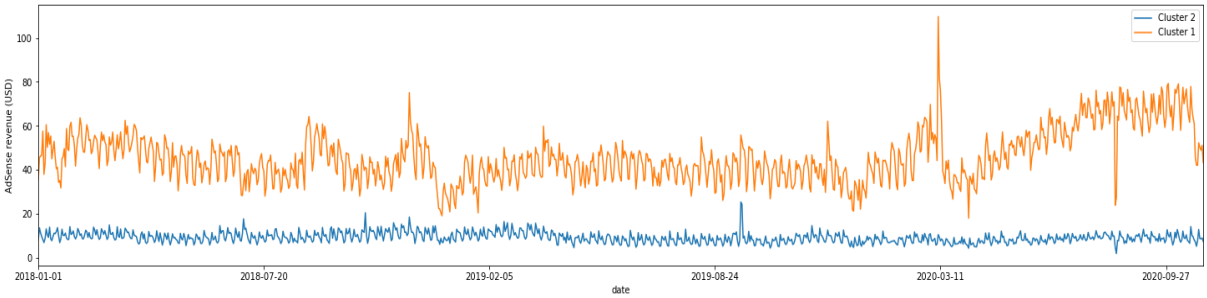

Figure 16: K-Means clustering with DTW - Euclidean barycenter AdSense revenue time series - without Cluster 2

I.3 Cluster 1 - Category Count

category	viewid
Science_and_Education	3
Vehicles	3
Home_and_Garden	2
Computers_Electronics_and_Technology	1
E-commerce_and_Shopping	1
Heavy_Industry_and_Engineering	1
Jobs_and_Career	1
Travel_and_Tourism	1

Table 10: Cluster 1 - Category Count

I.4 Cluster 1 - Country Count

country	viewid
United States of America	11
Germany	1
Italy	1

Table 11: Cluster 1 - Country Count

I.5 Cluster 2 - Category Count

category	viewid
Computers_Electronics_and_Technology	4
Health	4
Reference_Materials	3
E-commerce_and_Shopping	2
Sports	2
Business_and_Consumer_Services	1
Finance	1
Gambling	1
Games	1
Hobbies_and_Leisure	1
Home_and_Garden	1
Lifestyle	1
Science_and_Education	1
Vehicles	1

Table 12: Cluster 2 - Category Count

I.6 Cluster 2 - Country Count

country	viewid
United States of America	17
Netherlands	3
Germany	2
United Kingdom of Great Britain and Northern I...	2

Table 13: Cluster 2 - Country Count

I.7 Cluster 3 - Category Count

category	viewid
Food_and_Drink	1
Reference_Materials	1
Science_and_Education	1
Sports	1
Vehicles	1

Table 14: Cluster 3 - Category Count

I.8 Cluster 3 - Country Count

country	viewid
United States of America	3
Belgium	1
United Kingdom of Great Britain and Northern I...	1

Table 15: Cluster 3 - Country Count